Meltdown

- 针对微架构进行攻击的攻击方法（现在已经被修复）
  - 虽然我们不知道对应CPU的微架构是什么，但是我们可以猜啊。
- 用了Intel branch prediction不会检查对应权限的这个特性，以及利用cache实际上在性能上是不透明的特性，进行攻击，
- 得到branch的结果，实际上是一个队列，所以之前需要有复杂的耗时指令制造时间差。

攻击流程

- 首先，你知道对应的kernel address（假设你有无限的时间，那么这个假设还是合理的，攻击者总是很闲的，这也是为什么内核会随机找地方放数据之类的。）
  - 这个攻击仅仅针对，kernel 和User的地址放在一起的页表。（这是一种防止页表切换，提升系统性能的操作。）
- 清空cache（flash and reload）然后取对应的地址（有个细节，他是0和4096，这个是为了预防CPU的预取）
- 一次搞一个bit来（看buf的加载速度来判断是哪个地址）
  - 如果一次搞多个，`buffer`需要是`2^N`再乘以`4096`，我们可能没有足够的内存来一次读`N bit`。或许你可以一次读`8`个`bit`，然后`buffer`大小是`256*4096`。论文中有相关的讨论。
  - 还有就是程序运行的主要消耗是flush and reload，
所以一次读取一个`bit`，那么读取一个字节只需要`16`次`Reload`，一次读取一个字节，那么需要`256`次	`Reload`。**Why?**
- 用户进程是可以接管Pagefault handler的，所以不会被杀掉。

```
//
// Meltdown, including Flush+Reload
//


1   char buf[8192]
2
3   // the Flush of Flush+Reload
4   clflush buf[0]
5   clflush buf[4096]
6
7   <some expensive instruction like divide>
8
9   r1 = <a kernel virtual address>
10  r2 = *r1
11  r2 = r2 & 1       // speculated
12  r2 = r2 * 4096    // speculated
13  r3 = buf[r2]      // speculated
14
15  <handle the page fault from "r2 = *r1">
16
17  // the Reload of Flush+Reload
18  a = rdtsc
19  r0 = buf[0]
20  b = rdtsc
21  r1 = buf[4096]
22  c = rdtsc
23  if b-a < c-b:
24    low bit was probably a 1
```